

## MODULE – 6 LECTURE NOTES – 2

### IMAGE PROCESSING USING MATLAB

#### 1. Introduction

MATLAB (MATrix LABoratory) integrates computation, visualization, and programming in an easy-to-use environment. MATLAB provides comprehensive collections of MATLAB functions (M-files) called toolboxes, each of them are useful for particular class of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, image processing and many others.

Image processing tool box has extensive functions for many operations for image restoration, enhancement and information extraction. Some of the basic features of the image processing tool box are explained in this lecture. The applications are demonstrated using IRS LISS-III data covering a portion of the Uttara Kannada district in Karnataka.

Throughout this lecture the MATLAB command is given in pink color and the output is presented in blue.

#### 2. Images in MATLAB

MATLAB stores most images as two-dimensional arrays, in which each element of the matrix corresponds to a single pixel in the displayed image.

For example, an image composed of 200 rows and 300 columns of different colored dots would be stored in MATLAB as a 200-by-300 matrix. Some images, such as RGB, require a three-dimensional array, where the first plane in the third dimension represents the red pixel intensities, the second plane represents the green pixel intensities, and the third plane represents the blue pixel intensities.

This convention makes working with images in MATLAB similar to working with any other type of matrix data, and renders the full power of MATLAB available for image processing applications. For example, a single pixel can be selected from an image matrix using normal matrix subscripting.

**I(2,15)**

This command returns the value of the pixel at row 2, column 15 of the image.

MATLAB supports the following graphics file formats:

- BMP (Microsoft Windows Bitmap)
- HDF (Hierarchical Data Format)
- JPEG (Joint Photographic Experts Group)
- PCX (Paintbrush)
- PNG (Portable Network Graphics)
- TIFF (Tagged Image File Format)
- XWD (X Window Dump)

## 2.1 Pixel Coordinates

Generally, the most convenient method for expressing locations in an image is to use pixel coordinates. In this coordinate system, the image is treated as a grid of discrete elements, ordered from top to bottom and left to right, as illustrated in Fig. 1.

For pixel coordinates, the first component  $r$  (the row) increases downward, while the second component  $c$  (the column) increases to the right. Pixel coordinates are integer values and range between 1 and the length of the row or column.

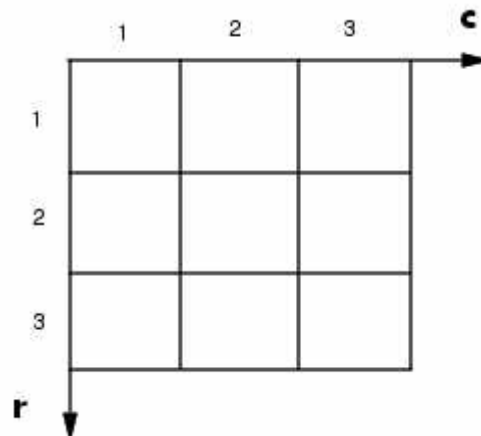


Fig.1. Pixel coordinates

### 3. Basic operations with MATLAB image processing tool box

#### 3.1 Read and Display an Image:

IRS LISS-III Band 4 image, which is a JPG file, is named here as image4.JPG. The following commands are used to display the image.

Clear the MATLAB workspace of any variables and close the open figure windows.

To read an image, use the *imread* command. Let's read in a JPEG image named image4. JPG, and store it in an array named I.

```
I = imread ('image4. JPG');
```

Now call *imshow* to display I.

```
imshow (I)
```

Image is displayed as shown in Fig 2. The image shows a part of the Uttara Kannada District, Karnataka. Some features in the image are

- (i) Arabian Sea on the left (as seen by the dark tone)
- (ii) Kalinadi in top half (represented by the linear feature in dark tone)
- (iii) Dense vegetation (the brighter tones in the image)
- (iv) Small white patches in the image are clouds.

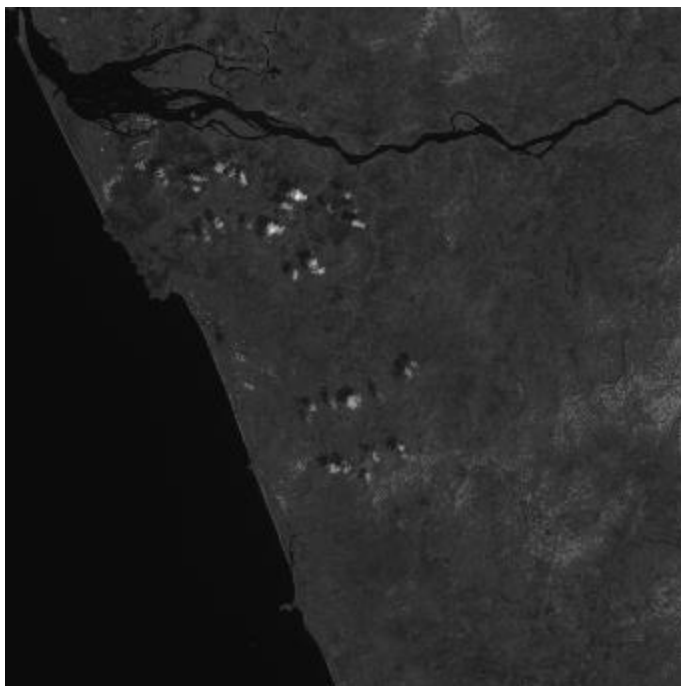


Fig. 2. Image displayed using  
*imshow*

The command *whos* is used to see how the variable *I* (which is the image here) is stored in memory.

*whos*

MATLAB responds to the command with the following output.

Name	Size	Bytes	Class
I	342x342	116964	uint8

Grand total is 116964 elements using 116964 bytes

The image is stored as a 2 dimensional matrix with 342 rows and 342 columns. Each element is saved as an unsigned 8-bit data.

### 3.2 Converting Image Storage Classes

Using MATLAB commands, it is possible to convert the data type in which the image is stored. For example, *uint8* (unsigned integer, 8 bit) and *uint16* (unsigned integer, 16 bit) data can be converted to double precision using the MATLAB function, *double*. However, converting between storage classes changes the way MATLAB and the toolbox interpret the image data. If it is desired to interpret the resulting array properly as image data, the original data should be rescaled or offset to suit the conversion.

For easier conversion of storage classes, use one of these toolbox functions: *im2double*, *im2uint8*, and *im2uint16*. These functions automatically handle the rescaling and offsetting of the original data.

For example, the following command converts a double-precision RGB (Red Green Blue) image with data in the range [0,1] to a *uint8* RGB image with data in the range [0,255].

*RGB2 = im2uint8(RGB1);*

### 3.3 Converting Graphics File Formats

MATLAB commands can also be used to convert the images saved in one format to another. To change the graphics format of an image, use *imread* to read the image and then save the image with *imwrite*, specifying the appropriate format.

For example, to convert an image from a BMP to a PNG, read the BMP image using *imread*, convert the storage class if necessary, and then write the image using *imwrite*, with 'PNG' specified as your target format.

```
bitmap = imread('image4.BMP','bmp');  
  
imwrite(bitmap,'image4.png','png');
```

## 4. Image Arithmetic

Standard arithmetic operations, such as addition, subtraction, multiplication, and division, when implemented on images are generally called Image Arithmetic. Image arithmetic has many uses in image processing both as a preliminary step and in more complex operations. It can be used to enhance or suppress the information, to detect the differences between two or more images of the same scene etc.

### 4.1 Adding Images

To add two images or add a constant value to an image, use the *imadd* function. *imadd* adds the value of each pixel in one of the input images with the corresponding pixel in the other input image and returns the sum in the corresponding pixel of the output image.

For example, the following commands use the image addition to superimpose one image on top of another. The images must be of the same size and class.

```
I = imread('image3.JPG');  
  
J = imread('image4.JPG');  
  
K = imadd(I,J); imshow(K)
```

In this example, image3.JPG and image4.JPG are IRS LISS-III Band-3 (Red) and Band-4 (Near Infrared) images, respectively. Added image is shown in Fig. 3.

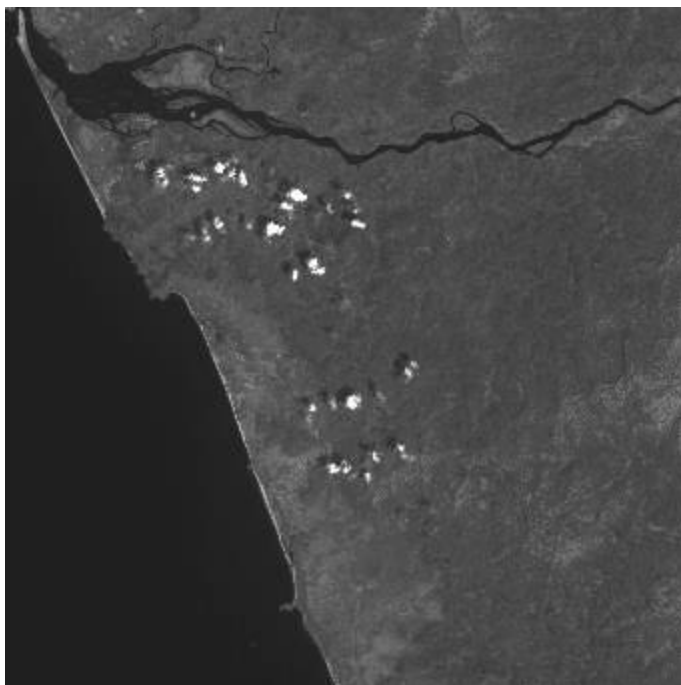


Fig. 3. Image after adding two images

One can also use addition to brighten an image by adding a constant value to each pixel.

For example, the following code brightens image4.JPG by adding a value 50 to all the pixel values.

```
I = imread('image4.JPG');  
J = imadd(I,50);
```

## 4.2 Subtracting Images

To subtract one image from another, or subtract a constant value from an image, use the *imsubtract* function. *imsubtract* subtracts each pixel value in one of the input images from the corresponding pixel in the other input image and returns the result in the corresponding pixel in an output image.

```
X= imread('image5.JPG');  
J= imread('image4.JPG');  
K= imssubtract(X,J);
```

## 4.3 Multiplying Images

MATLAB command *immultiply* does an element-by-element multiplication of each corresponding pixel in a pair of input images and returns the product of these multiplications in the corresponding pixel in an output image.

Image multiplication by a constant, referred to as scaling, is a common image processing operation. When used with a scaling factor greater than one, scaling increases the brightness of an image; a factor less than one reduces the brightness of an image. Image scaling preserves the relative contrast of the image and hence produces a much more natural brightening/ darkening effect.

For example, the following commands are used for rescaling or to brighten the image4.JPG by using a factor 3 and to display the brightened image.

```
I = imread('image4.JPG');  
J = immultiply(I,3.0);  
figure, imshow(J);
```

Fig. 4 shows the display of the rescaled image.

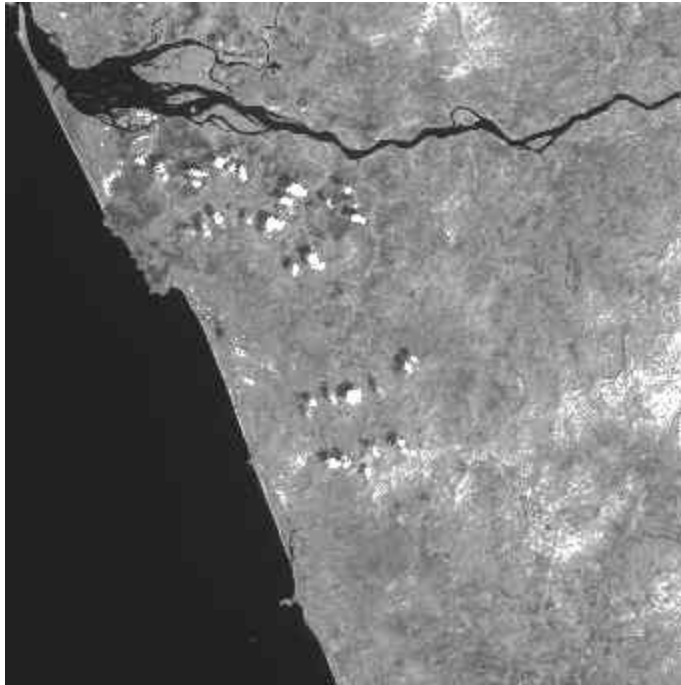


Fig. 4. Image multiplied by an integer 3

#### 4.4 Dividing Images

The *imdivide* function in the image processing toolbox is used for an element-by-element division of each corresponding pixels in a pair of input images and to return the result in the corresponding pixel in an output image.

Image division is also called ratioing.

Image division, like image subtraction, can be used to detect changes in two images. However, instead of giving the absolute change for each pixel, division gives the fractional change or ratio between corresponding pixel values.

### 5. Special Display Techniques

The command *imshow* displays an image. Additional functions are available in the MATLAB image processing toolbox to exercise more direct control over the display format. Adding a color bar, image resizing, image rotation and image cropping are a few options used to perform specialized display of an image.

## 5.1 Adding a Colorbar

The *colorbar* function can be used to add a color bar to an axes object. A color bar added to an axes object that contains an image object indicates the data values that the different colors or intensities in the image correspond to as shown in Fig. 5. The MATLAB commands for image display with color bar are given below.

```
F = imread('image5.JPG');  
  
imshow(F), colorbar
```

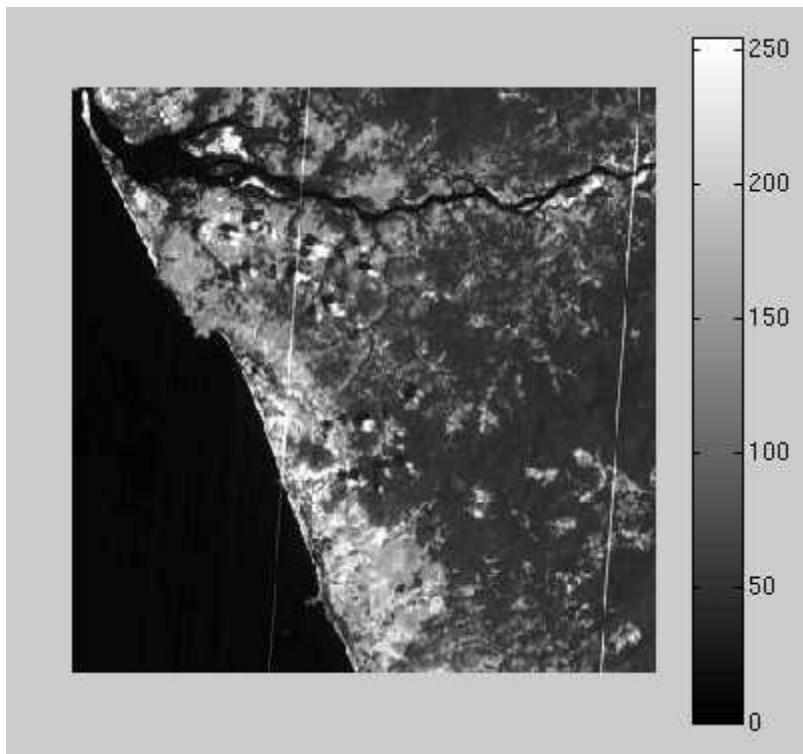


Fig.5.  
Image with color bar

In Fig. 5 are IRS LISS-III Band-5 image is displayed. The image is displayed in gray scale. A colorbar for the image is shown alongside the image indicating the values corresponding to different gray levels.

## 5.2 Image Resizing

An image can be resized using the *imresize* function. The function takes two primary arguments viz., (i) The image to be resized and (ii) The magnification factor as given below.

```
F = imread ('image5.JPG');  
  
J = imresize(F,0.5);
```



The first function reads the IRS LISS-III Band-5 image and saves it as F. The second function is used to resize the image 0.5 times and to save the output as J.

Another way of using the *imresize* function is by specifying the actual size of the output image, instead of the resize factor. The command below creates an output image of size 100-by-150.

```
Y = imresize (F, [100 150] );
```

### 5.3 Image Rotation

To rotate an image, the *imrotate* function can be used. The function requires two primary arguments as input. These are (i) The image to be rotated and (ii) The rotation angle. The rotation angle should be specified in degrees. For a positive value, *imrotate* rotates the image counterclockwise; and for a negative value, *imrotate* rotates the image clockwise. The *imrotate* option in matlab allows the choice of interpolation method to be as either 'nearest', 'bilinear' or 'bicubic'. They are popular interpolation techniques in which bicubic interpolation can produce pixel values outside the original image.

For example, the following set of commands may be used to read the IRS LISS-III Band-5 image and to rotate the same through 35 degrees counterclockwise. The rotated image is stored as J and is displayed at the end as shown in Fig. 6.

```
F = imread('image5.JPG');
```

```
J = imrotate (F,35,'bilinear');
```

```
figure, imshow(J)
```

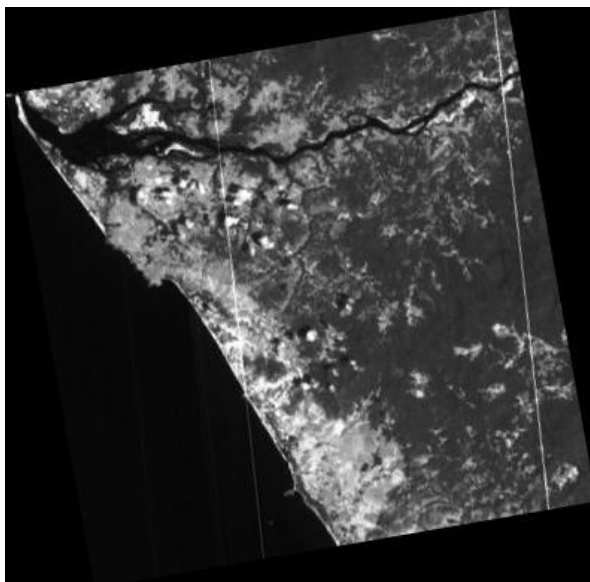


Fig. 6.  
Image rotated by 35 degrees

## 5.4 Image Cropping

To subset or to crop a rectangular portion of an image, the *imcrop* function can be used. The function requires two arguments as input viz., (i) The image to be cropped and (ii) The coordinates of a rectangle that defines the crop area.

The coordinates of the rectangle may be specified manually or can be selected from the image display window.

If *imcrop* is called without specifying the coordinates of the rectangle, the cursor changes to a cross hair when it is over the image. Click on one corner of the region to be selected and while holding down the mouse button, drag across the image towards the diagonally opposite corner of the required rectangle. Thus a rectangle is drawn around the selected area. When the mouse button is released, *imcrop* extracts the corresponding coordinates and creates a new image of the selected region.

## 6. Image Analysis

A range of standard image processing operations for image analysis are also supported by the MATLAB image processing toolbox. Two categories of operations available for image analysis and image enhancement are mentioned here:

- Extracting pixel values and statistics
- Analyzing images to extract information about their essential structure e.g contours, and edges

### 6.1 Extracting pixel values

The toolbox includes two functions that provide information about the pixel values or the color data values of an image viz., *pixval* and *impixel*.

The function *pixval* interactively displays the data values for pixels as the cursor is moved over the image. It can also display the Euclidean distance between two pixels.

The second function, *impixel*, on the other hand, returns the data values for a selected pixel or set of pixels. The coordinates of the pixel is used as the input argument. The coordinates can be specified manually.

If the coordinates are not specified manually, they are selected using the cursor as it moves over the image. Below are the set of commands used to extract the pixel values.

```
imshow image4.JPG;
```

```
vals = impixel;
```

## 6.2 Extracting Summary Statistics

Standard statistics of an image such as mean, standard deviation, correlation coefficient etc. can also be computed using the functions available in the MATLAB image processing toolbox.

For example, the functions *mean2* and *std2* compute the mean and standard deviation of the elements of the image, which is stored as a matrix. The function *corr2* computes the correlation coefficient between two matrices of the same size.

## 6.3 Image Contours

Similar to the contour function in MATLAB, the toolbox function *imcontour* can be used to display the contour plot of the data in an image. Contours connect pixels of equal pixel values. The *imcontour* function automatically sets up the axes so their orientation and aspect ratio match the image.

For example, the following set of commands is used to read an image and to display the image information in the form of contours.

```
I = imread('image5.JPG');
```

```
figure, imcontour(I)
```

This reads the IRS LISS-III Band-5 image (which is shown in Fig. 5) and generates the contours within the image. The contour image displayed is shown in Fig. 7.

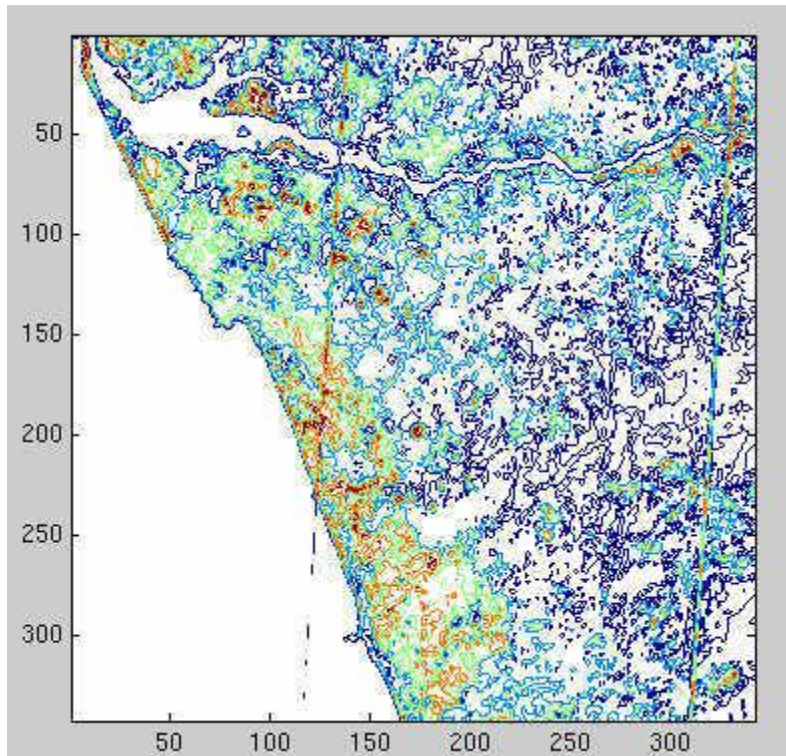


Fig. 7.  
Contour plot of an  
Image

## 6.4 Edge Detection

Edges are the places in an image corresponding to object boundaries. Therefore, in an image edges generally correspond to rapid changes in the intensities or pixel values. The toolbox function *edge* looks for places in the image where the intensity changes rapidly and hence detects the edges. Function *edge* returns a binary image containing 1's where edges are found and 0's elsewhere.

Any one of the following criteria is used to detect the rapid change in the intensity.

- i Places where the first derivative of the intensity is larger in magnitude than some threshold
- ii Places where the second derivative of the intensity is zero

For some of these estimators, it can be specified whether the operation should be sensitive to horizontal or vertical edges, or both.

MATLAB toolbox contains more than one method for the edge detection eg., Sobel method and Canny's method.

The *edge* function takes two arguments viz., the image for which the edges are to be identified and the edge detection method.

For example, the following commands are used to detect the edges in the image5.JPG using Canny's methods. The edges are displayed as shown in Fig. 8

```
F = imread('image5.JPG');  
BW1 = edge(F,'canny');  
figure, imshow(BW1)
```

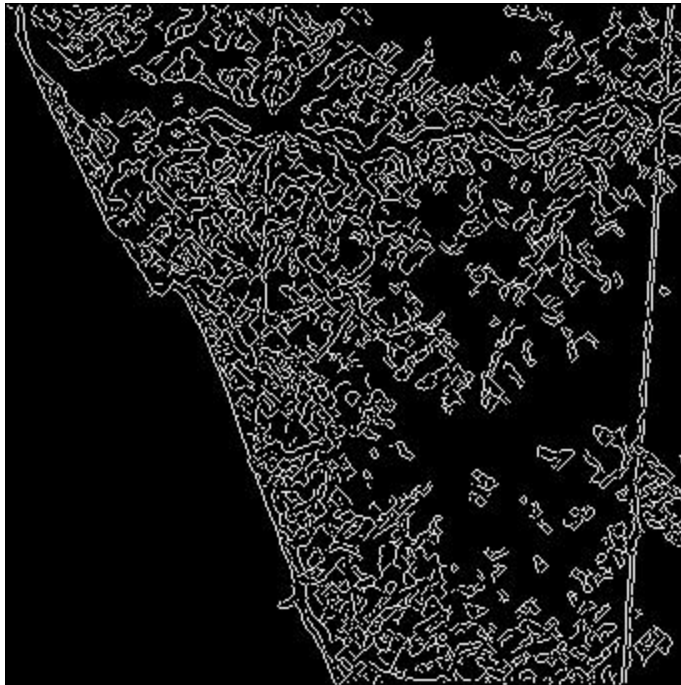


Fig.8.  
Edge detection Image

Canny's method is the most powerful edge-detection method, as it uses two different thresholds to detect strong and weak edges separately. Further the weak edges connected to strong edges are displayed in the output. This method is therefore less likely than the others to be "fooled" by noise, and more likely to detect true weak edges.

## 7. Image Enhancement

Image enhancement techniques are used to improve an image or to enhance the information contained in the image. For example, enhancement of the signal-to-noise ratio, enhancement of certain features by modifying the colors or intensities so than they can be easily identified or differentiated from others.

### 7.1 Intensity Adjustment

Intensity adjustment is a technique for mapping an image's intensity values to a new range.

For example, as seen from Fig. 2, contrast of the original display of image4.JPG is poor. Although the pixels can be displayed in the intensity range of 0-255 (in the 8-bit system), only a narrow range is used for the display.

To see the distribution of intensities in image4.JPG in its current state, a histogram can be created by calling the *imhist* function. (Precede the call to *imhist* with the figure command so that the histogram does not overwrite the display of the image in the current figure window.)

figure, imhist (I)

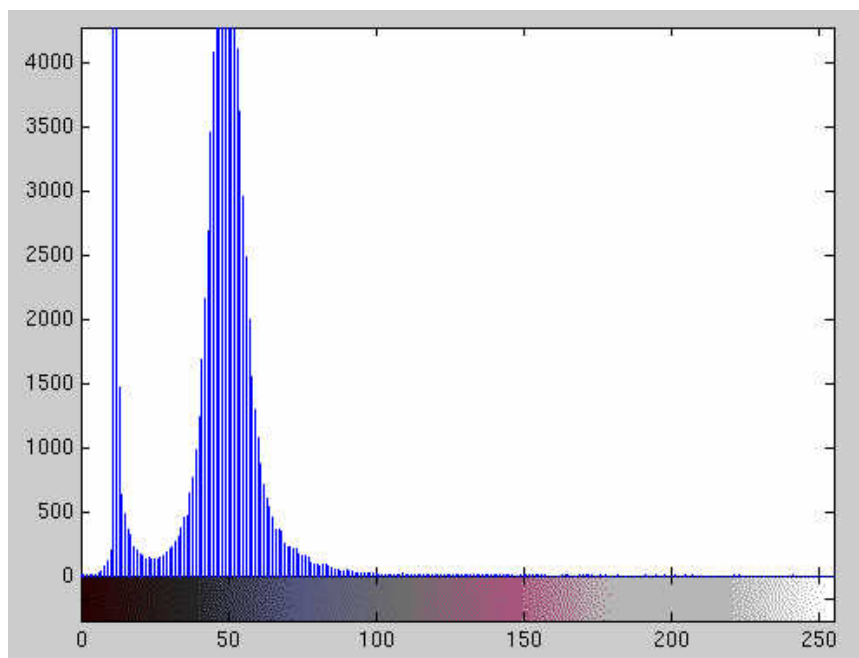


Fig.9. Histogram of raw image

From the histogram it can be seen that most of the values are concentrated in the region 10-80. There are very few values above 80. On the other hand the display levels are equally distributed for the entire range, which results in poor contrast in the image display. Contrast in the image can be improved if the data values are remapped to fill the entire intensity range [0,255].

This kind of adjustment can be achieved with the *imadjust* function. The function takes 3 arguments as input viz., the image to be adjusted, the input intensity range and the corresponding output intensity range.

The general syntax of *imadjust* is

```
J = imadjust (I, [low_in high_in], [low_out high_out] )
```

Where, low\_in and high\_in are the intensities in the input image, which are mapped to low\_out, and high\_out in the output image.

For example, the code below performs the adjustment described above.

```
I = imread('image4.JPG');  
J = imadjust (I, [0.0 0.3], [0 1]);
```

The first vector passed to *imadjust*, [0.0 0.3], specifies the low and high intensity values of the input image *I*. The second vector, [0 1], specifies the scale over which you want to map them in the output image *J*. Thus, the example maps the intensity value 0.0 in the input image to 0 in the output image, and 0.3 to 1.

Note that one must specify the intensities as values between 0 and 1 regardless of the class of *I*. If *I* uses unsigned 8-bit data format (*uint8*), the values supplied are multiplied by 255 to determine the actual values to use.

## 7.2 Histogram Equalization

Another approach for image enhancement is through histogram equalization. In histogram equalization, for each input intensity level, corresponding output intensity levels are assigned based on the frequency. Higher frequency ranges in the input images generally carry more information compared to the lower frequency ranges. Hence, by assigning more display levels to the higher frequency ranges, information enhancement is better achieved through histogram equalization compared to the simple linear intensity enhancement.



Histogram equalization is done in MATLAB using the command *histeq*. The command spreads the intensity values of *I* (original image) over the full range, thereby improving the contrast. Store the modified image in the variable *I2*.

```
I2 = histeq (I);
```

New equalized image, *I2*, is displayed in a new figure window using the command *imshow*.

```
figure, imshow(I2)
```

Fig. 10 shows the display of the imaged enhanced through histogram equalization.

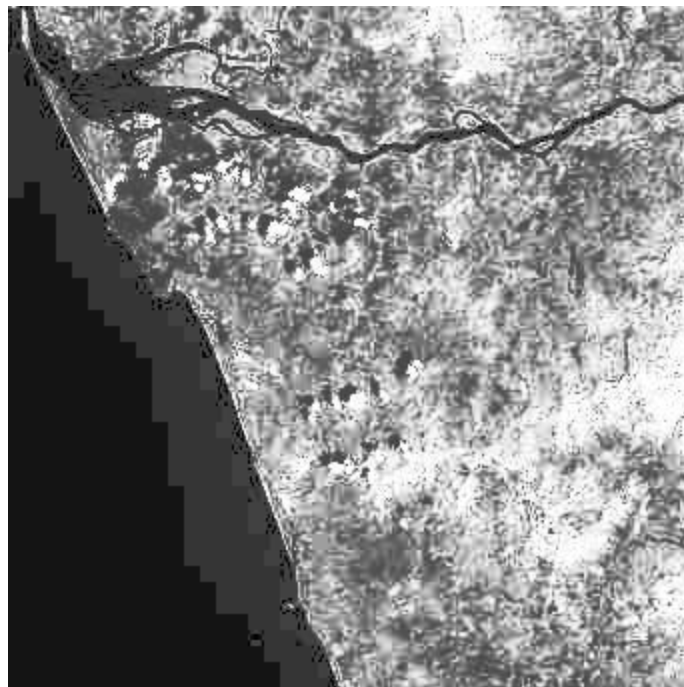


Fig. 10. Histogram equalized Image

The new image *I2* can be written back to disk using the *imwrite* command. If it is to be saved as a PNG file, use *imwrite* and specify a filename that includes the extension 'png'.

```
imwrite (I2, 'image4.png')
```

The contents of the newly written file can be checked using *imfinfo* function to see what was written to disk.

```
imfinfo('image4.png')
```

As a matter of fact MATLAB image processing tool box has many more capabilities and only a small portion of them is explained in this lecture.



## **Bibliography**

1. MathWorks Inc., *Image Processing Tool Box Users Guide*